SQLT Usage Instructions (Doc ID 1614107.1)

215187.1 SQLTXPLAIN (SQLT) 12.1.06 January 30, 2014 Tool that helps to diagnose SQL statements performing poorly

SQLT Overview

SQLTXPLAIN, also known as **SQLT**, is a tool provided by Oracle Server Technologies Center of Expertise - ST CoE. **SQLT** main methods input one SQL statement and output a set of diagnostics files. These files are commonly used to diagnose SQL statements performing poorly or those which generate wrong results.

Once **installed**, you can use **SQLT** to analyze a SQL statement by passing its text within a script (including bind variables), or by providing its **SQL_ID**. The SQL_ID can be found in AWR and ASH reports and the HASH_VALUE in SQL_TRACE output (above the SQL text and identified by the "hv=" token). You can also find these columns in the V$SQL view. Please refer to the following document for example selects:

> **Document 1627387.1** How to Determine the SQL_ID for a SQL Statement

SQLT main methods connect to the database and collect execution plans, Cost-based Optimizer CBO statistics, schema objects metadata, performance statistics, configuration parameters, and other elements that influence the performance of the one SQL being analyzed. These methods produce a set of data for the SQL_ID in question including a "main" report which displays information in html format. You can find some suggestions about how you might be able to use the SQLT main report in the following document:

> **Document 1922234.1** SQLT Main Report: Usage Suggestions

SQLT may use the Oracle Diagnostic and/or the Oracle Tuning Packs if your site has a license for them. These two provide enhanced functionality to the **SQLT** tool. During **SQLT** installation you can specify if one of these two packages is licensed by your site. If none, **SQLT** still provides some basic information that can be used for initial SQL diagnostics.

Security Model

During the **installation** of this **SQLT** tool, 2 users and one role are created. These users and role have fixed names.

The **SQLT** repository is owned by user **SQLTXPLAIN**. **SQLT** users need to provide the **SQLTXPLAIN** password everytime they use any of the **main methods**. User **SQLTXPLAIN** is granted the following system privileges:

- **CREATE SESSION**

- **CREATE TABLE**

The **SQLT** set of PL/SQL packages and Views is owned by user **SQLTXADMIN**. This **SQLTXADMIN** user is locked and identified by a random password. **SQLTXADMIN** is granted the following system privileges:

- **ADMINISTER SQL MANAGEMENT OBJECT**
- **ADMINISTER SQL TUNING SET**
- **ADVISOR**
- **ALTER SESSION**
- **ANALYZE ANY**
- **SELECT ANY DICTIONARY**
- **SELECT_CATALOG_ROLE**

All **SQLT** users must be granted the **SQLT_USER_ROLE** before they can use any of the **main methods**. This **SQLT_USER_ROLE** role is granted the following system privileges:

- **ADVISOR**
- **SELECT_CATALOG_ROLE**

It's not possible to use SYS as **SQLT** user by default in 12c due to a change in the security model for PL/SQL. To address this change **SQLTXADMIN** needs to be granded **INHERIT PRIVILEGES** on SYS.
GRANT INHERIT PRIVILEGES ON USER SYS TO SQLTXADMIN
You can find more details in Oracle® Database PL/SQL Language Reference 12c Release 1 (12.1) - Invoker's Rights and Definer's Rights (AUTHID Property) and in Oracle® Database Security Guide 12c Release 1 (12.1) - Managing Security for Definer's Rights and Invoker's Rights

---

Installing SQLT

SQLT installs under its own schemas **SQLTXPLAIN** and **SQLTXADMIN**. It does not install any objects into the application schema(s). You can install this version of **SQLT** in Oracle databases 10.2, 11.1, 11.2 and higher, on UNIX, Linux or Windows platforms.

Installation steps:

1. **Uninstall a prior version (optional).**

   This optional step removes all obsolete **SQLTXPLAIN**/**SQLTXADMIN** schema objects and prepares the environment for a fresh install. Skip this step if you want to preserve the existing content of the **SQLT** repository (recommended).

   ```
   # cd sqlt/install
   # sqlplus / as sysdba
   SQL> START sqdrop.sql
   ```

2. **Execute installation script `sqlt/install/sqcreate.sql` connected as `SYS`.**

   ```
   # cd sqlt/install
   # sqlplus / as sysdba
   SQL> START sqcreate.sql
   ```

During the installation you will be asked to enter values for these parameters:

1. **Optional Connect Identifier (mandatory when installing in a Pluggable Database)**

   In some restricted-access systems you may need to specify a connect identifier like **@PROD**. If a connect identifier is not needed, enter nothing and just hit the "Enter" key. Entering nothing is the most common setup.
   The Connect Identifier is a mandatory parameter when installing SQLT in a Pluggable Database.

2. **SQLTXPLAIN password.**

   Case sensitive in most systems.

3. **SQLTXPLAIN Default Tablespace.**

   Select from a list of available permanent tablespaces which one should be used by **SQLTXPLAIN** for the **SQLT** repository. It must have more than 50MB of free space.

4. **SQLTXPLAIN Temporary Tablespace.**

   Select from a list of available temporary tablespaces which one should be used by **SQLTXPLAIN** for volatile operations and objects.

5. **Optional Application User.**

   This is the user that issued the SQL statement to be analyzed. For example, if this were an EBS system specify **APPS**, on Siebel you would specify **SIEBEL** and on People Soft **SYSADM**. You won't be asked to enter the password for this user. You can add additional **SQLT** users after the tool is installed, by granting them role **SQLT_USER_ROLE**.

6. **Licensed Oracle Pack. (T, D or N)**

   You can specify **T** for Oracle Tuning, **D** for Oracle Diagnostic or **N** for none. If **T** or **D** is selected, **SQLT** may include licensed content within the diagnostics files it produces. Default is **T**. If **N** is selected, **SQLT** installs with limited functionality.

If a silent installation is desired, there are three options to pass all 6 installation parameters:

1. **In a file.**

   Executing first a script with pre-defined values, similar to sample script **sqlt/install/sqdefparams.sql**. Then use **sqlt/install/sqcsilent.sql** instead of **sqlt/install/sqcreate.sql**.

```
# cd sqlt/install
# sqlplus / as sysdba
SQL> START sqdefparams.sql
SQL> START sqcsilent.sql
```

2. **In-line.**

   Executing **sqlt/install/sqcsilent2.sql** instead of **sqlt/install/sqcreate.sql**. The former inputs the same 6 installation parameters but in-line.

```
# cd sqlt/install
# sqlplus / as sysdba
SQL> START sqcsilent2.sql '' sqltxplain USERS TEMP '' T
```

3. **Internal installation at Oracle.**

   Executing **sqlt/install/sqcinternal.sql** instead of **sqlt/install/sqcreate.sql**. The former executes **sqlt/install/sqdefparams.sql** followed by **sqlt/install/sqcsilent.sql**.

```
# cd sqlt/install
# sqlplus / as sysdba
SQL> START sqcinternal.sql
```

If you need further help with install issues, you can get help in the following community thread:

SQLTXPLAIN: SQLT Installation Issues

Uninstalling SQLT

Uninstalling **SQLT** removes the **SQLT** repository and all **SQLTXPLAIN/SQLTXADMIN** schema objects. **SQLTXPLAIN** and **SQLTXADMIN** users also gets dropped. To uninstall **SQLT** simply execute **sqlt/install/sqdrop.sql** connected as **SYS**.

```
# cd sqlt/install
# sqlplus / as sysdba
SQL> START sqdrop.sql
```

Upgrading SQLT

If you have a prior version of **SQLT** already installed in your system, you can upgrade **SQLT** to its latest version while partially preserving most objects of your existing **SQLT** repository. The new migrated **SQLT** repository can then be used to restore CBO statistics or to perform a **COMPARE** between old and new executions of **SQLT**.

To upgrade **SQLT**, simply do an **installation** without performing the optional **uninstall** step.

If the upgrade fails, then it is possible the prior **SQLT** version was too old to be upgraded. In such case please proceed to **uninstall** **SQLT** first, followed by a clean **installation**.

Frequently Asked Questions

Refer to MOS Doc ID:**1454160.1**.

Main Methods

Before attempting using any of the **SQLT** main methods, be sure **SQLT** has been **installed** and that your **SQLT** user has been granted the **SQLT_USER_ROLE**.

If SQLT has been installed bypassing SQL*Net (meaning that you did not input any value for the Connect Identifier during the installation), then prior to running any of the SQLT main methods from a remote client, you will need to manually set the connect_identifier parameter. Ie. if you connected using sqlplus scott/tiger@myprod then you will need to execute: EXEC sqltxadmin.sqlt$a.set_sess_param('connect_identifier', '@myprod');

SQLT provides 7 main methods that generate diagnostics details for one SQL statement: **XTRACT**, **XECUTE**, **XTRXEC**, **XTRSBY**, **XPLAIN**, **XPREXT** and **XPREXC**. While **XTRACT**, **XECUTE**, **XTRXEC**, **XTRSBY**, **XPREXT** and **XPREXC** handle bind variables and understand about bind peeking, **XPLAIN** does not. This is because **XPLAIN** is based on the **EXPLAIN PLAN FOR** command which is blind to bind peeking. For this reason avoid using **XPLAIN** if possible.

Besides the bind peeking limitation on **XPLAIN** all 7 main methods provide enough diagnostics details to make an initial assessment of a SQL performing poorly or generating wrong results. If the SQL still resides in memory or in the Automatic Workload Repository (**AWR**) use then **XTRACT** or **XTRXEC**, else use **XECUTE**. For Data Guard or standby read-only databases use **XTRSBY**. Use **XPLAIN** only if the other methods are not feasible. **XPREXT** and **XPREXC** are similar to **XTRACT** and **XECUTE** but they disable some **SQLT** features in order to improve **SQLT** performance.

XTRACT Method

Use this method if you know the **SQL_ID** or the **HASH_VALUE** of the SQL to be analyzed, else use **XECUTE**.

The **SQL_ID** can be found on an **AWR** report, and the **HASH_VALUE** on any SQL Trace (above the SQL text and identified by the "hv=" token).

If the SQL is still in memory, or it has been captured by **AWR**, then XPREXT finds it and provides a set of diagnostics files, else XPREXT errors out.

Important performance statistics, like actual number of rows per execution plan operation, will be available if the SQL was parsed while parameter **STATISTICS_LEVEL** was set to **ALL** when the SQL was hard-parsed. You can also produce same valuable performance statistics by including the following CBO hint in your SQL: **/*+ GATHER_PLAN_STATISTICS */**. On 11g you may want your SQL to contain the following CBO Hints for enhanced diagnostics: **/*+ GATHER_PLAN_STATISTICS MONITOR */**

When this method is used, it asks for the **SQLTXPLAIN** password, which is needed to export the **SQLT** repository corresponding to this execution of XPREXT.

This method requires the application user executing **SQLT** to be granted the **SQLT_USER_ROLE** role.

To use this XPREXT method, be sure **SQLT** has been **installed** first, then connect into SQL*Plus as the application user that executed the SQL to be analyzed and execute the **sqlt/run/sqltxtract.sql** script passing the **SQL_ID** or **HASH_VALUE**.

```
# cd sqlt/run
# sqlplus apps
SQL> START sqltxtract.sql [SQL_ID]|[HASH_VALUE] [sqltxplain_password]
SQL> START sqltxtract.sql 0w6uydn50g8cx sqltxplain_password
SQL> START sqltxtract.sql 2524255098 sqltxplain_password
```

XECUTE Method

This method provides more detail than **XTRACT**. As the name XECUTE implies, it executes the SQL being analyzed, then it produces a set of diagnostics files. Its major drawback is that if the SQL being analyzed takes long to execute, this method will also take long.

As a rule of thumb, use this method only if the SQL takes less than 1hr to execute, else use **XTRACT**.

Before you can use this XECUTE method, you have to create a text file that contains your SQL text. If the SQL includes bind variables, your file must contain the bind variable declaration and assignment. Use **sqlt/input/sample/script1.sql** as an example. Your SQL should contain the token **/* ^^unique_id */** which should be spelled out exactly as it shows here. In other words: please do not change it.

If your SQL requires binds with data types not allowed by SQL*Plus, or if it uses collections, you may be restricted to embed your SQL into an anonymous PL/SQL block. In such case use **sqlt/input/sample/plsql1.sql** as an input example to this method.

For statements that modify data, i.e. INSERT/UPDATE/DELETE, a savepoint is created prior to statement execution and the transaction is rolled back to the savepoint at the conclusion of the session. For further information regarding SAVEPOINT please consult the Oracle Concepts reference manual.

When this method is used, it asks for the **SQLTXPLAIN** password, which is needed to export the **SQLT** repository corresponding to this execution of XECUTE.

This method requires the application user executing **SQLT** to be granted the **SQLT_USER_ROLE** role.

To use the XECUTE method, be sure **SQLT** has been **installed** first, then connect into SQL*Plus as the application user that executed the SQL to be analyzed and execute the **sqlt/run/sqltxecute.sql** script passing the name of the text file that contains your SQL text and its bind variables. You may want to place this file into the **sqlt/input** directory and run XECUTE while standing on the sqlt main directory, as shown below.

```
# cd sqlt
# sqlplus apps
SQL> START [path]sqltxecute.sql [path]scriptname [sqltxplain_password]
```

```
SQL> START run/sqltxecute.sql input/sample/script1.sql sqltxplain_password
```

XTRXEC Method

This method combines the features of **XTRACT** and **XECUTE**. Actually, XTRXEC executes both methods serially. The **XTRACT** phase generates a script that contains the extracted SQL together with the binds declaration and assignment for an expensive plan found for the requested SQL statement. XTRXEC then executes the **XECUTE** phase using the script created by the first.

The selection of the values of the bind variables used by **XTRACT** to create the script is based on the peeked values at the moment the most expensive plans in memory were generated. Expensive plans are selected according to their average elapsed time.

If XTRXEC errors out having executed only the 1st phase (**XTRACT**), you may need to review the script used during the 2nd phase (**XECUTE**) and adjust the bind variables accordingly. This is specially true when uncommon data types are used.

When this method is used, it asks for the **SQLTXPLAIN** password, which is needed to export the **SQLT** repository corresponding to this execution of XTRXEC.

This method requires the application user executing **SQLT** to be granted the **SQLT_USER_ROLE** role.

To use this XTRXEC method, be sure **SQLT** has been **installed** first, then connect into SQL*Plus as the application user that executed the SQL to be analyzed and execute the **sqlt/run/sqltxtrxec.sql** script passing the **SQL_ID** or **HASH_VALUE**.

```
# cd sqlt/run
# sqlplus apps
SQL> START sqltxtrxec.sql [SQL_ID]|[HASH_VALUE] [sqltxplain_password]
SQL> START sqltxtrxec.sql 0w6uydn50g8cx sqltxplain_password
SQL> START sqltxtrxec.sql 2524255098 sqltxplain_password
```

XTRSBY Method

Use this method if you need to analyze a SQL executed on a Data Guard or stand-by read-only database. You need to know the **SQL_ID** or the **HASH_VALUE** of the SQL to be analyzed.

Create on Primary database a link to read-only database connecting as any user that has access to the data dictionary. A DBA account would be fine:

```
CREATE PUBLIC DATABASE LINK V1123 CONNECT TO mydba IDENTIFIED by mydba_password
USING '(DESCRIPTION = (ADDRESS=(PROTOCOL=TCP)
(HOST=coesrv14.us.oracle.com)(PORT=1521))(CONNECT_DATA=(SID = V1123)))';
```

If the SQL is still in memory in the read-only database, then XTRSBY finds it and provides a set of diagnostics files, else XTRSBY errors out.

Important performance statistics, like actual number of rows per execution plan operation, will be available if the SQL was parsed while parameter **STATISTICS_LEVEL** was set to **ALL** when the SQL was hard-parsed in the read-only database. You can also produce same valuable performance statistics by including the following CBO hint in your SQL: **/*+ GATHER_PLAN_STATISTICS */**. On 11g you may want your SQL to contain the following CBO Hints for enhanced diagnostics: **/*+ GATHER_PLAN_STATISTICS MONITOR */**

When this method is used, it asks for the **SQLTXPLAIN** password, which is needed to export the **SQLT** repository corresponding to this execution of XTRSBY.

XTRSBY takes 3 parameters: the SQL id, the DB_LINK id, and the **SQLTXPLAIN** password.

This method requires the application user executing **SQLT** to be granted the **SQLT_USER_ROLE** role.

To use this XTRSBY method, be sure **SQLT** has been **installed** on the Primary first, and replicated into the read-only database. Then connect into SQL*Plus in Primary and execute the **sqlt/run/sqltxtrsby.sql** script passing the **SQL_ID** or **HASH_VALUE** followed by the **DB_LINK**.

```
# cd sqlt/run
# sqlplus apps
SQL> START sqltxtrsby.sql [SQL_ID]|[HASH_VALUE] [sqltxplain_password] [DB_
SQL> START sqltxtrsby.sql 0w6uydn50g8cx sqltxplain_password V1123
SQL> START sqltxtrsby.sql 2524255098 sqltxplain_password v1123
```

In addition to XTRSBY you may want to execute **sqlt/utl/sqlhc.sql** or **sqlt/utl/sqlhcxec.sql** directly from the read-only database. These two read-only scripts do not install anything on the database nor they execute DML commands. They provide additional information that is not available in XTRSBY.

XPLAIN Method

This method is based on the **EXPLAIN PLAN FOR** command, therefore it is blind to bind variables referenced by your SQL statement. Use this method only if **XTRACT** or **XECUTE** are not possible.

Before using the XPLAIN method, you have to create a text file that contains your SQL text. If the SQL includes bind variables, you have two options: leave the SQL text "as is", or carefully replace the binds with literals of the same datatype. Use **sqlt/input/sample/sql1.sql** as an example.

When this method is used, it asks for the **SQLTXPLAIN** password, which is needed to export the **SQLT** repository corresponding to this execution of XPLAIN.

This method requires the application user executing **SQLT** to be granted the **SQLT_USER_ROLE** role.

To use this XPLAIN method, be sure **SQLT** has been **installed** first, then connect into SQL*Plus as the application user that executed the SQL to be analyzed and execute the **sqlt/run/sqltxplain.sql** script passing the name of the text file that contains your SQL text. You may want to place this file into the sqlt/input directory and run XPLAIN while standing on the sqlt main directory, as shown below.

```
# cd sqlt
# sqlplus apps
SQL> START [path]sqltxplain.sql [path]filename [sqltxplain_password]
SQL> START run/sqltxplain.sql input/sample/sql1.sql sqltxplain_password
```

XPREXT Method

Use this method if you have used **XTRACT** and you need a faster execution of **SQLT** while disabling some **SQLT** features. Script **sqlt/run/sqltcommon11.sql** shows which features are disabled.

Use this method if you know the **SQL_ID** or the **HASH_VALUE** of the SQL to be analyzed, else use **XPREXC**. The **SQL_ID** can be found on an **AWR** report, and the **HASH_VALUE** on any SQL Trace (above the SQL text and identified by the "hv=" token).

When this method is used, it asks for the **SQLTXPLAIN** password, which is needed to export the **SQLT** repository corresponding to this execution of XPREXT.

This method requires the application user executing **SQLT** to be granted the **SQLT_USER_ROLE** role.

To use this XPREXT method, be sure **SQLT** has been **installed** first, then connect into SQL*Plus as the application user that executed the SQL to be analyzed and execute the **sqlt/run/sqltxprext.sql** script passing the **SQL_ID** or **HASH_VALUE**.

```
# cd sqlt/run
```

```
# sqlplus apps
SQL> START sqltxprext.sql [SQL_ID]|[HASH_VALUE] [sqltxplain_password]
SQL> START sqltxprext.sql 0w6uydn50g8cx sqltxplain_password
SQL> START sqltxprext.sql 2524255098 sqltxplain_password
```

XPREXC Method

Use this method if you have used **XECUTE** and you need a faster execution of **SQLT** while disabling some **SQLT** features. Script **sqlt/run/sqltcommon11.sql** shows which features are disabled.

As a rule of thumb, use this method only if the SQL takes less than 1hr to execute, else use **XPREXT**.

Before you can use this XPREXC method, you have to create a text file that contains your SQL text. If the SQL includes bind variables, your file must contain the bind variable declaration and assignment. Use **sqlt/input/sample/script1.sql** as an example. Your SQL should contain the token **/* ^^unique_id */** which should be spelled out exactly as it shows here. In other words: please do not change it.

If your SQL requires binds with data types not allowed by SQL*Plus, or if it uses collections, you may be restricted to embed your SQL into an anonymous PL/SQL block. In such case use **sqlt/input/sample/plsql1.sql** as an input example to this method.

For statements that modify data, i.e. INSERT/UPDATE/DELETE, a savepoint is created prior to statement execution and the transaction is rolled back to the savepoint at the conclusion of the session. For further information regarding SAVEPOINT please consult the Oracle Concepts reference manual.

When this method is used, it asks for the **SQLTXPLAIN** password, which is needed to export the **SQLT** repository corresponding to this execution of XPREXC.

This method requires the application user executing **SQLT** to be granted the **SQLT_USER_ROLE** role.

To use the XPREXC method, be sure **SQLT** has been **installed** first, then connect into SQL*Plus as the application user that executed the SQL to be analyzed and execute the **sqlt/run/>sqltxprexc.sql** script passing the name of the text file that contains your SQL text and its bind variables. You may want to place this file into the **sqlt/input** directory and run XPREXC while standing on the sqlt main directory, as shown below.

```
# cd sqlt
# sqlplus apps
SQL> START [path]sqltxprexc.sql [path]scriptname [sqltxplain_password]
SQL> START run/sqltxprexc.sql input/sample/script1.sql sqltxplain_passwor
```

Special Methods

Besides the **main methods** **SQLT** provides some special methods.

The most popular special method is **COMPARE**. This method takes as input two prior executions of **SQLT** (any of the **main methods**) and produces a report with a gap analysis.

The other special methods are: **TRCANLZR**, **TRCAXTR**, **TRCASPLIT** and **XTRSET**. The first three act on a SQL Trace and the latter on a set of SQL statements.

COMPARE Method

Use this COMPARE method when you have two similar systems (SOURCES) and the same SQL statement performs fine in one of them but not in the other. This method helps to pin-point the differences between the two SOURCES in terms of plans, metadata, CBO statistics, initialization parameters and bug fix-control. **SQLT** has to be **installed** first in both, and any of the **main methods** must have been used on the same SQL in

both systems.

The compare can take place on any of the two SOURCES or on a 3rd COMPARE system. The latter should contain the **SQLT** repositories of the two SOURCES. To import a **SQLT** repository use the syntax provided in the sqlt_99999_readme.html file generated by any of the **main methods**.

Once the COMPARE system contains the repositories from both SOURCES, execute **sqlt/run/sqltcompare.sql** connecting as **SYS** or the application user. A list of **STATEMENT_ID** is presented, from which you choose which two **SQLT** stored executions you want to compare. Once you indicate the two **STATEMENT_ID** you are asked next for particular **PLAN_HASH_VALUE** from both SOURCES.

```
# cd sqlt
# sqlplus sqltxplain
SQL> START [path]sqltcompare.sql [STATEMENT_ID 1] [STATEMENT_ID 2]
SQL> START run/sqltcompare.sql 92263 72597
SQL> START run/sqltcompare.sql
```

TRCANLZR Method

This method takes as input a SQL Trace filename and proceeds to analyze this file. The actual trace must be located in the **TRCA$INPUT1** directory, which defaults to the **USER_DUMP_DEST** directory during **installation**.

The TRCANLZR method also has the capability of analyzing simultaneously several related traces, treating them as a set. This functionality is needed when analyzing parallel execution PX traces. In such case, create a **control.txt** file with a list of traces (one filename per line and without path specification) and place this **control.txt** into the **TRCA$INPUT1** or **TRCA$INPUT2** directories. These two directories default to **USER_DUMP_DEST** and **BACKGROUND_DUMP_DEST** respectively during **installation**. TRCANLZR will then read the **control.txt** file from one of the two input directories, and it will find the set of traces in either one of those two directories.

TRCANLZR is like **TKPROF** but with extended functionality. When it analyzes a trace or a set of traces, it also includes schema object characteristics like CBO statistics and some other valuable performance metrics.

To use this TRCANLZR method, be sure **SQLT** has been **installed** first. Then, start SQL*Plus connecting as the application user that generated the trace and execute the **sqlt/run/sqltrcanlzr.sql** script passing the name of the trace to be analyzed, or the name of the **control.txt** file populated with filenames. Do not include any path specification.

```
# cd sqlt
# sqlplus [application_user]
SQL> START [path]sqltrcanlzr.sql [SQL Trace filename|control.txt]
SQL> START run/sqltrcanlzr.sql V1122_ora_24292.trc
SQL> START run/sqltrcanlzr.sql control.txt
```

TRCAXTR Method

This method does the same than **TRCANLZR** but when the trace analysis completes, it continues with a **XTRACT** for the Top SQL found in the trace. Basically it consolidates all the generated reports by **TRCANLZR** and **XTRACT** on the Top SQL.

To use this TRCAXTR method, be sure **SQLT** has been **installed** first. Then, navigate to the **sqlt/run** directory and start SQL*Plus connecting as the application user that generated the trace. From there, execute the **sqlt/run/sqltrcaxtr.sql** script passing the name of the trace to be analyzed, or the name of the **control.txt** file populated with filenames. Do not include any path specification.

```
# cd sqlt/run
# sqlplus [application_user]
```

```
SQL> START sqltrcaxtr.sql [SQL Trace filename|control.txt]
SQL> START sqltrcaxtr.sql V1122_ora_24292.trc
SQL> START sqltrcaxtr.sql control.txt
```

TRCASPLIT Method

This method takes as input a SQL Trace filename which was created by EVENT 10046 and some other EVENT(s) (usually 10053). Then it proceeds to split this input trace file into two output files. One with the trace lines corresponding to EVENT 10046 and the other with its complement. In other words, the second file contains those trace lines that are not part of the EVENT 10046 grammar. So if the input trace was created using simultaneously EVENT 10046 and EVENT 10053, the resulting output files would be the 10046 trace and the 10053 trace. The actual input trace must be located in the **TRCA$INPUT1** directory, which defaults to the **USER_DUMP_DEST** directory during **installation**.

To use this TRCASPLIT method, be sure **SQLT** has been **installed** first. Then, start SQL*Plus connecting as any **SQLT** user and execute the **sqlt/run/sqltrcasplit.sql** script passing the name of the trace to be splitted. Do not include any path specification.

```
# cd sqlt
# sqlplus [sqlt_user]
SQL> START [path]sqltrcasplit.sql [SQL Trace filename]
SQL> START run/sqltrcasplit.sql V1122_ora_24292.trc
```

XTRSET Method

The XTRSET extracts from memory or **AWR** a list of SQL statements identified by their **SQL_ID** or **HASH_VALUE** then it executes the **XTRACT** on each of the SQL statements. At the end it consolidates all the **SQLT** files into a single compressed file. This XTRSET is used when benchmarking the same set of SQL statements over a series of tests.

When this method is used, it asks once for the **SQLTXPLAIN** password, which is needed to export the **SQLT** repository for each execution of **XTRACT** on the list of SQL statements.

To use this XTRSET method, **SQLT** has to be **installed** first. Navigate to the **sqlt/run** directory and start SQL*Plus connecting as the application user that issued all or most of the SQL statements. From there, execute the **sqlt/run/sqltxtrset.sql** script. When asked, pass the comma-separated list of SQL statements identified by their **SQL_ID** or **HASH_VALUE**, and the password for **SQLTXPLAIN**.

```
# cd sqlt/run
# sqlplus [application_user]
SQL> START sqltxtrset.sql
List of SQL_IDs or HASH_VALUEs: 2yas208zgt5cv, 6rczmqdtg99mu, 8w8tjgac6tv1
```

Advanced Methods and Modules

SQLT provides some additional functionality beyond the **main methods** and the **special methods**. Use these advanced methods and modules only if requested by Oracle Support: **PROFILE**, **XGRAM**, **XPLORE** and **XHUME**. The latter is for exclusive use of Oracle Support and only in an internal testing environment.

PROFILE Method

This PROFILE method allows a quick fix on 10g when a SQL performing poorly happens to have a known better-performing plan. This better plan can be in memory in the same or different system, or in **AWR** in the

same or different system. In other words, if a better plan is available, this method allows "pinning" this plan using a custom SQL Profile. Before you use this method you have to use any of the **main methods** on the SQL for which you want to extract and pin its plan. On 11g or higher you want to use SQL Plan Management (SPM) instead of this method.

Be aware that PROFILE uses a **DBMS_SQLTUNE** API, which is part of the SQL Tuning Advisor, therefore licensed through the Oracle Tuning pack. Use this PROFILE method only if your site has a license for the Oracle Tuning pack.

To use this PROFILE method, be sure **SQLT** has been **installed** and used in the SOURCE system, then connect into SQL*Plus as **SYS** or **SQLTXPLAIN** and execute the **sqlt/utl/sqltprofile.sql** script. It will ask for the **STATEMENT_ID** out of a list of prior **SQLT** executions. After a **STATEMENT_ID** is selected, it will ask for a **PLAN_HASH_VALUE** out of a list of available plans. These plans were captured and stored by **SQLT** when **XTRACT** or **XECUTE** were used on the SQL of concern.

SQLT does not have to be installed in the TARGET system where the custom SQL Profile is implemented.

There are basically 4 steps in this PROFILE method.

1. **Use XTRACT or XECUTE on the SOURCE system.**
2. **Execute `sqlt/utl/sqltprofile.sql` in SOURCE to generate a script with the custom SQL Profile.**
3. **Review the generated script and adjust the SQL text if needed. For example, to remove the comment caused by the `/* ^^unique_id */` if XECUTE was used.**
4. **Execute the generated script in the TARGET system where the plan will be pinned.**

```
# cd sqlt/utl
# sqlplus sqltxplain
SQL> START sqltprofile.sql [statement id] [plan hash value];
SQL> START sqltprofile.sql 32263 923669362;
SQL> START sqltprofile.sql 32263;
SQL> START sqltprofile.sql;
```

The custom SQL Profile created by this method is based on the plan outline data and not in scaling factors, therefore it is more steady. If you later want to drop this custom SQL Profile, you can find the drop command within the script that PROFILE generated.

If you don't have **SQLT** installed in the SOURCE system, or you cannot execute **XTRACT** or **XECUTE** for the SQL of concern, you can achieve the same functionality offered by the PROFILE method by using **sqlt/utl/coe_xfr_sql_profile.sql** instead. This script also uses **DBMS_SQLTUNE**; therefore a license for the Oracle Tuning pack is required.

If your system is 11g and you are considering using this PROFILE method, review the dynamic readme generated by any of the **main methods** and look for "Create SQL Plan Baseline from SQL Set". You may want to consider using SQL Plan Management SPM through a SQL Set as documented in the dynamic readme.

XGRAM Module

The XGRAM module provides functionality to modify CBO Histograms either to enhance CBO Statistics for some columns or as part of a Test Case. With this module you can insert, update or delete Histograms or individual Buckets.

Alphabetical list of scripts that implement the XGRAM module:

1. **`sqlt/utl/xgram/sqlt_delete_column_hgrm.sql`**
2. **`sqlt/utl/xgram/sqlt_delete_hgrm_bucket.sql`**
3. **`sqlt/utl/xgram/sqlt_delete_schema_hgrm.sql`**
4. **`sqlt/utl/xgram/sqlt_delete_table_hgrm.sql`**
5. **`sqlt/utl/xgram/sqlt_display_column_stats.sql`**
6. **`sqlt/utl/xgram/sqlt_insert_hgrm_bucket.sql`**

7. `sqlt/utl/xgram/sqlt_set_bucket_size.sql`
8. `sqlt/utl/xgram/sqlt_set_column_hgrm.sql`
9. `sqlt/utl/xgram/sqlt_set_min_max_values.sql`

The XGRAM module is installed automatically when **SQLT** is installed. If you ever need to install and use this XGRAM module outside **SQLT** you would only need to install one package and use the list of scripts above (removing **SQLTXADMIN** dependencies).

XPLORE Module

The XPLORE module helps when after a database upgrade a SQL starts to perform poorly or it may produce apparent wrong results. If switching **optimizer_features_enable** (OFE) to the database release prior to the upgrade the SQL performs fine again, or it produces different results, you can use this XPLORE module to try to identify which particular Optimizer feature or fix introduced the undesired behavior. Identifying the particular culprit can help to troubleshoot further or do more research on this particular feature and/or fix.

This module toggles initialization and fix control parameters to discover plans.

Use XPLORE only when ALL these conditions are met:

1. **SQL performs poorly or returns wrong results while using a "bad" plan.**
2. **The bad plan can be reproduced on a test system (no data is preferred).**
3. **A "good" plan can be reproduced on the test system by switching OFE.**
4. **You need to narrow reason to specific parameter or bug fix control.**
5. **You have full access to the test system, including `SYS` access.**

Do not use XPLORE when ANY of these conditions is true:

1. **The SQL statement may cause corruption or update data.**
2. **There is high volume of data in tables referenced by SQL.**
3. **The execution of the SQL may take longer than a few seconds.**

To install and use this XPLORE module, read corresponding **sqlt/utl/xplore/readme.txt**.

XHUME Module

This module is for the exclusive use of Oracle Support. It must be used only in an Oracle internal system. It updates the data dictionary and this operation is not supported by Oracle.

XHUME can be used to discover plans that are only possible with older version of schema object statistics related to one SQL. After a Test Case (**TC**) is created using **SQLT**, this XHUME module restores systematically prior versions of the statistics and generates plans by executing the SQL being studied. It captures the plan that each version of the statistics can generate. Then produces a report, which can be used to understand the plan instability, or to find an elusive plan that can be used to create a SQL Profile or SQL Plan Baseline.

This module should never be used in a Production system because it modifies the data dictionary. Use only on an Oracle internal Test enviornment.

As an alternative to modify the creation date of the Test Case (**TC**) schema objects, you can change the date on the server prior to the **TC** implementation, and reset to current date after the **TC** is created. This temporary prior date must be at least one month old, so all history of schema object statistics would have a save time newer than the **TC** object creation time.

Use XHUME only when ALL these conditions are met:

1. **SQL is known to generate more than one plan, and one or more perform poorly.**
2. **Bind peeking has been ruled out as the culprit of the plan instability.**
3. **Changes to CBO parameters has been ruled out as the culprit of the plan instability.**
4. **You have a `SQLT TC` where a known plan is generated (a "good" or a "bad" one).**

5. **You need to understand the plan instability, or you are looking for an elusive known "good" plan.**
6. **You have full access to the Oracle internal test system, including `SYS` access.**

Do not use XHUME when ANY of these conditions is true:

1. **The SQL statement may cause corruption or update data.**
2. **You only have a `TC` on a Production environment.**
3. **You do not have `TC` created with `SQLT`.**
4. **You do not have SYS access to the Oracle internal test system that contains your `TC`.**
5. **Bind peeking or CBO parameters have not been ruled out as culprit of the plan instability.**

To install and use this XHUME module, read corresponding **`sqlt/utl/xhume/readme.txt`**.

Discuss SQLT!

The window below is a live discussion of this article (not a screenshot). We encourage you to join the discussion by clicking the "Reply" link below for the entry you would like to provide feedback on. If you have questions or implementation issues with the information in the article above, please share that below.

Todos os Lugares > My Oracle Support Community > Oracle Database (MOSC) > SQL Performance (MOSC) > Discussõ

67 Respostas Última resposta: 20/07/2014 08:15 por **Shahabuddin**

Steve Dixon-Oracle 23/04/2014 09:16

SQLTXPLAIN (SQLT): General Discussion

Due to the wide variety of posts received on this thread, we decided to create a number of mor
Please use these new threads for the problem area you are encountering.

For general SQLT Discussions continue to use this thread

Targeted threads:

SQLTXPLAIN: SQLT Installation Issues.
SQTXPLAIN: Dealing with Long Execution Times
SQTXPLAIN: Dealing with Errors reported in SQLT MAIN report
SQTXPLAIN: Using SQLT on a Stand-by or Dataguard
SQTXPLAIN: Interpreting and Understanding SQLT Output

References:

Document 215187.1 SQLT (SQLTXPLAIN) - Tool that helps to diagnose SQL statements pe
Document 1454160.1 FAQ: SQLT (SQLTXPLAIN) Frequently Asked Questions
Document 1521607.1 Troubleshooting SQLT Issues

Please note that in addition to SQLT, the SQL Heathcheck Script can be used to check the env based Optimizer (CBO) statistics, schema object metadata, configuration parameters and othe being analyzed. You can find a discussion regarding this script in the following thread:

Discussion about the SQL Health-Check Script (SQLHC) as Featured in Notes:1366133.1 a

108212 Visualizações **Rótulos:**

Classificação de usuário médio **Minha Pontuação:**

(0 Classificações)

Gogala Mladen 23/05/2013 04:04 (em resposta a Steve Dixon-Oracle)
1. Re: Discussion about: Note:215187.1 SQLT (SQLTXPLAIN) - Tool that helps to diagnose SQL s

I love that tool, I am using it to format trace files from my DBA_Helper tool that I wrote a long time
http://mgogala.byethost5.com/dba_helper_1.0.30.zip

The output is much better than tkprof but it uses a lot of CPU.

Ações

sshaik145202 23/05/2013 04:04 (em resposta a Gogala Mladen)
2. Re: Discussion about: Note:215187.1 SQLT (SQLTXPLAIN) - Tool that helps to diagnose SQL s

SQLT is Performance tunings teams guided tool and I personally feel that's the best tool that we g with every release.

Off topic.. but you can also use sqlhc.sql if you want quick and easy understanding and troublesho go for SQLT.

REFERENCES

NOTE:1614201.1 - SQLT Changes